Introduction to the Abstract Meaning Representation (AMR)

http://tiny.cc/amrtutorial

http://amr.isi.edu/

Why abstract?

- English provides many ways to express even simple ideas.
    - Too many to simply write down a few rules to characterize, e.g., paraphrase alternations.
- For many NLP applications, we want to abstract away from the details of English grammar.
    - What is deeper than syntax? Semantics!

But hasn’t this been done before?

- Long tradition in linguistics and CL of formalizing semantics.
- The key insights behind AMR:

(1) statistical NLP needs a semantic representation that is **practical for large-scale human annotation** (sembanking)

  - What is practical? **limited canonicalization**

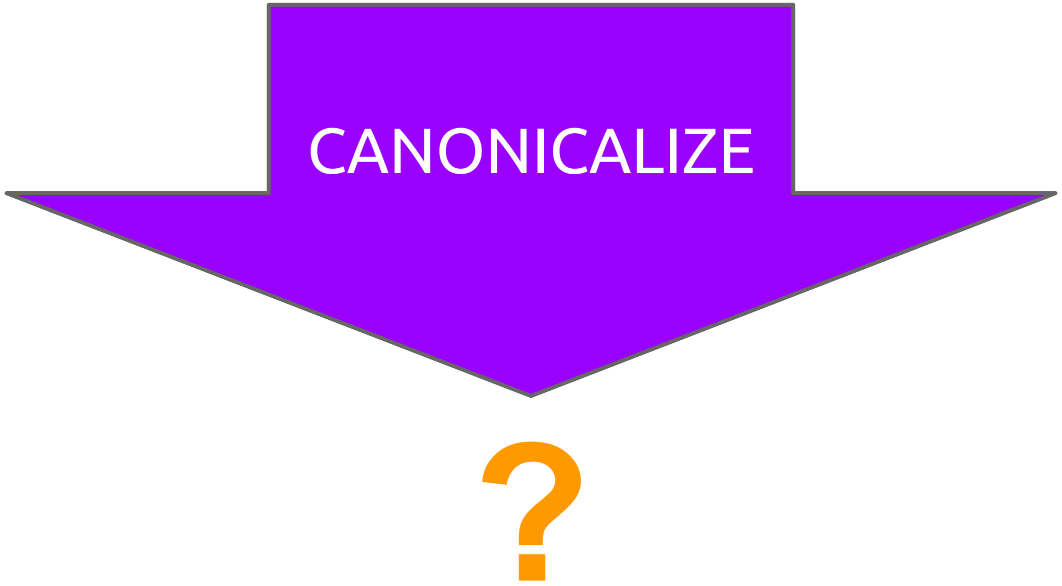
(2) many crucial aspects of meaning can be captured with broad coverage **in a single data structure**

The man described the mission as a disaster.

The man's description of the mission: disaster.

As the man described it, the mission was a disaster.

The man described the mission as disastrous.

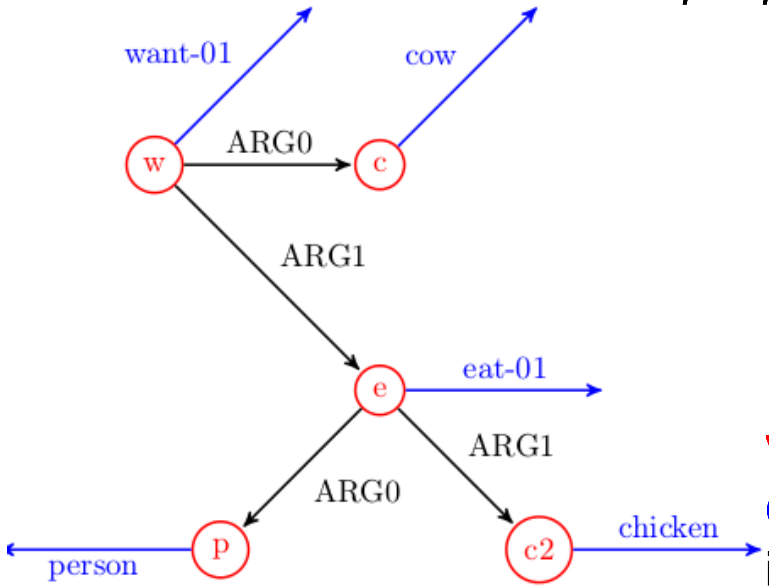


Roadmap for Part I of the tutorial

- Fundamentals of the representation
  - how AMR graphs are structured to represent concepts and relations
- Hands-on annotation practice
  - the annotation tool, simple examples
- Survey of linguistic/semantic phenomena
- Comparison to other representations
- Annotation practice
  - more realistic examples

AMR Notation Format (PENMAN)

"Cows want people to eat chicken."



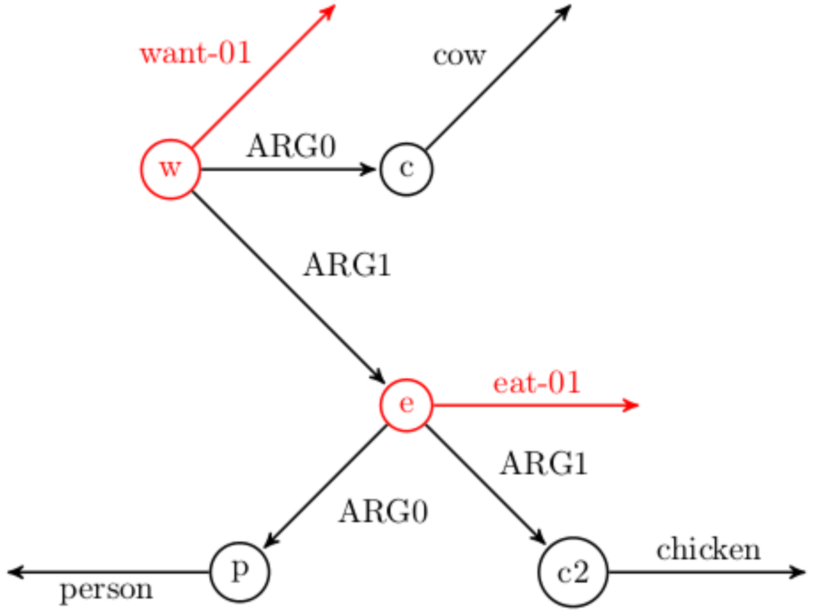
=

```
(w / want-01
      :ARG0 (c / cow)
      :ARG1 (e / eat-01
            :ARG0 (p / person)
            :ARG1 (c2 / chicken)))
```

Variables are the unique nodes.
Concepts show what the variable is an instance of.

Variables and Concepts

"Cows want people to eat chicken."



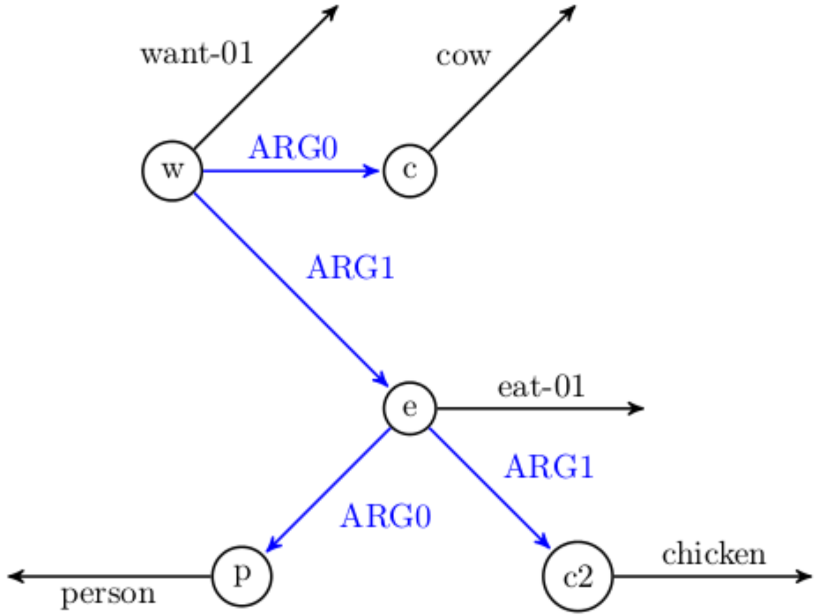
=

```
(w / want-01
    :ARG0 (c / cow)
    :ARG1 (e / eat-01
        :ARG0 (p / person)
        :ARG1 (c2 / chicken)))
```

Some of the nodes are sense disambiguated (the predicates), dropping tense and aspect.
Non-predicative, unnamed concepts are often left as stemmed words, dropping plurality

Relations

"Cows want people to eat chicken."



=

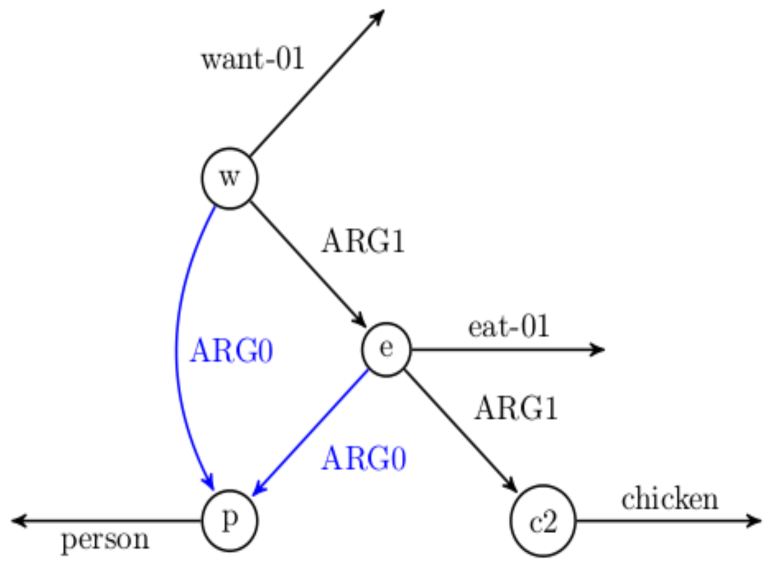
```
(w / want-01
      :ARG0 (c / cow)
      :ARG1 (e / eat-01
            :ARG0 (p / person)
            :ARG1 (c2 / chicken)))
```

Relations are preceded by colons, and can be numbered arguments or more general relations.

The general tendency is that ARG0 is the agent and ARG1 the undergoer.

Re-entrancies

“People want to eat chicken.”



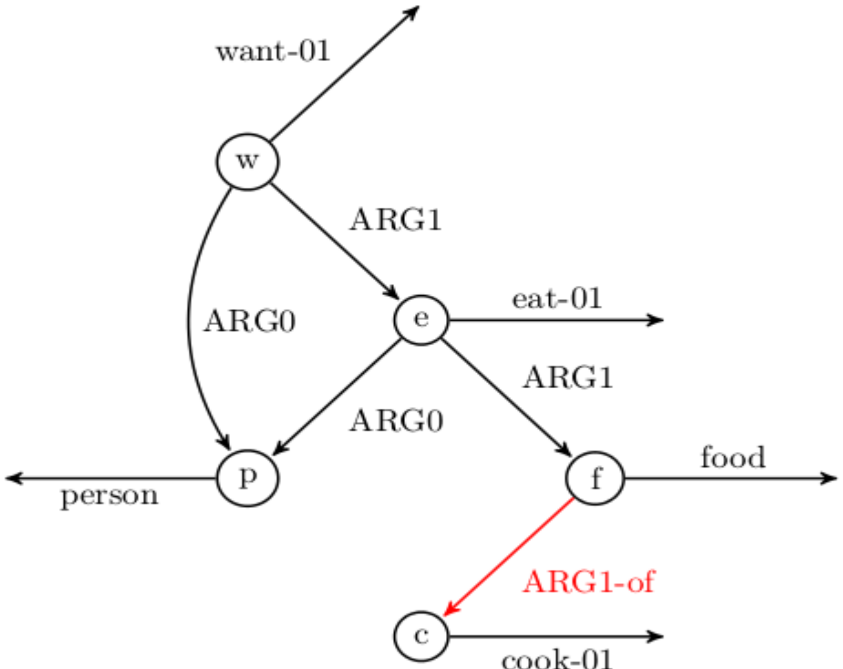
=

```
(w / want-01
     :ARG0 (p / person)
     :ARG1 (e / eat-01
          :ARG0 p
          :ARG1 (c2 / chicken)))
```

Concepts can have more than one semantic role, which are called “re-entrancies”.
We represent this by reusing variable names like p

Inverse-of relations

"People want to eat cooked food"



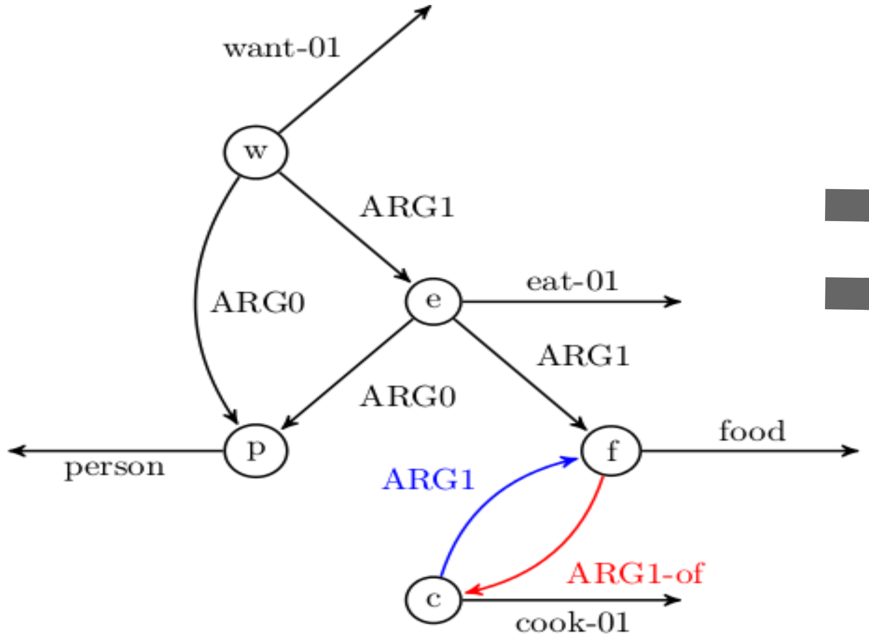
=

```
(w / want-01
      :ARG0 (p / person)
      :ARG1 (e / eat-01
            :ARG0 p
            :ARG1 (f / food
                  :ARG1-of (c / cook-01))))
```

For things like relative clauses, the predicate modifies the term using an **inverse** relation, arg1-**of**.

Inverse-of relations

"People want to eat cooked food"



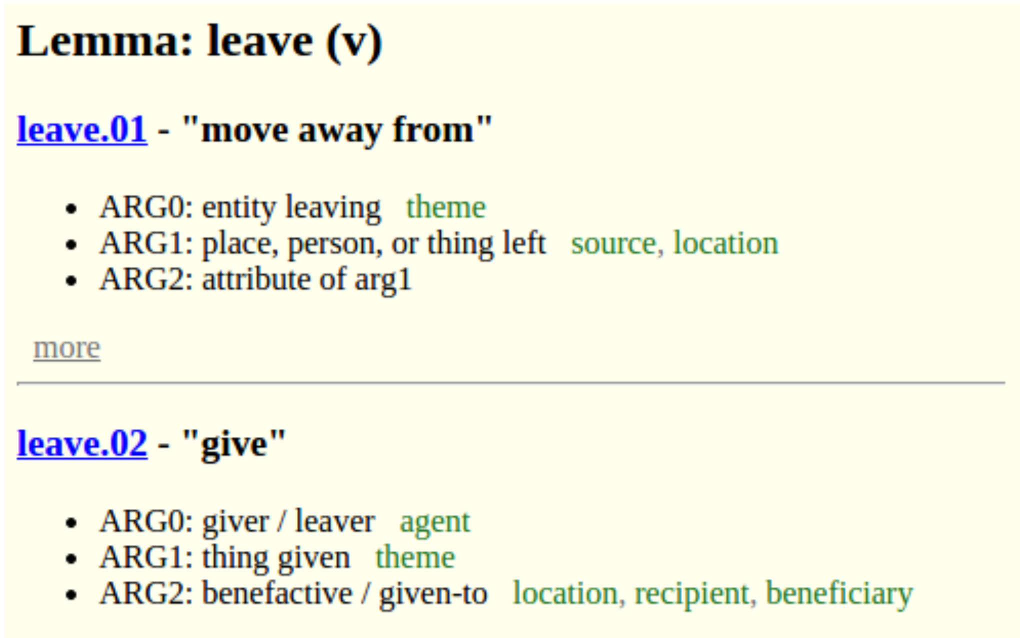
=

```
(w / want-01
      :ARG0 (p / person)
      :ARG1 (e / eat-01
            :ARG0 p
            :ARG1 (f / food
                  :ARG1-of (c / cook-01))))
```

These inverse relations could be viewed as just a notational trick - blue and red convey the same meaning.

Lexicon: what “want-01” and “:ARG0” mean

- AMR concepts are not merely the sentence tokens
- We use an inventory of conceptual frames for predicates: the unified PropBank rolesets.
- The numbered (core) semantic roles are specific to each roleset.



Lemma: leave (v)

leave.01 - **"move away from"**

- ARG0: entity leaving   theme
- ARG1: place, person, or thing left   source, location
- ARG2: attribute of arg1

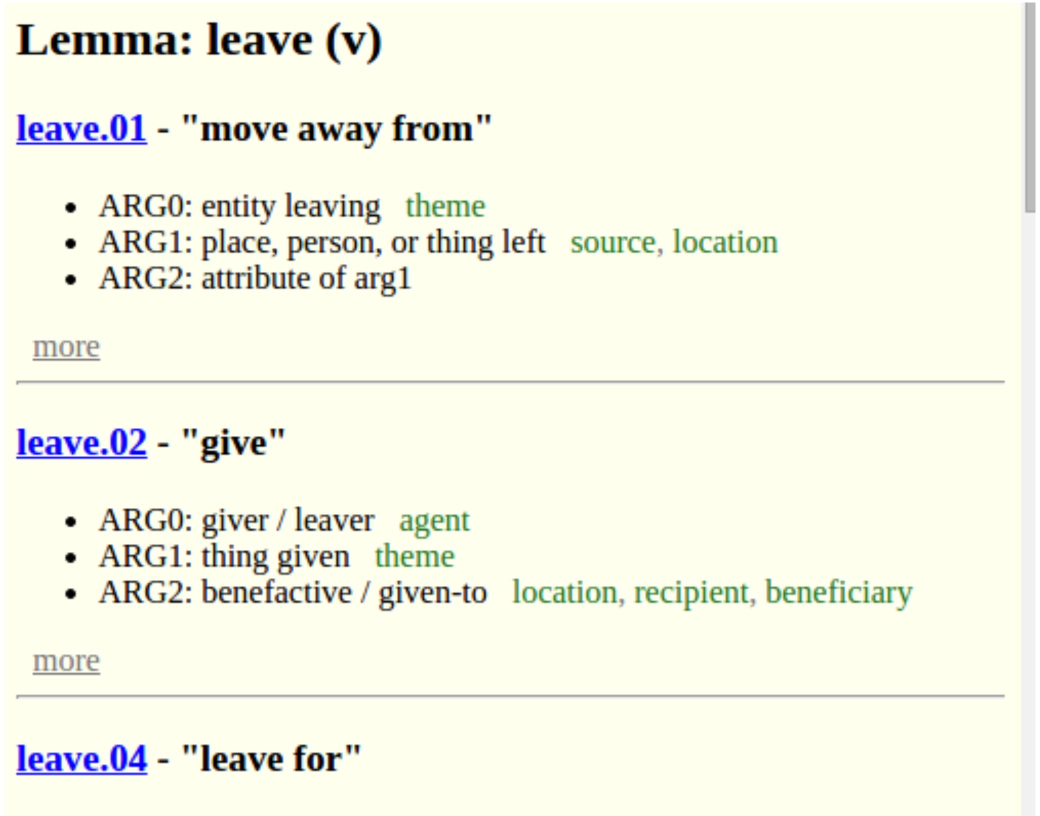
more

leave.02 - **"give"**

- ARG0: giver / leaver   agent
- ARG1: thing given   theme
- ARG2: benefactive / given-to   location, recipient, beneficiary

Lexicon: what “want-01” and “arg0” mean

Annotators see
a list that shows
all possible rolesets
and what each numbered
argument means.



Lemma: leave (v)

leave.01 - "move away from"

- ARG0: entity leaving theme
- ARG1: place, person, or thing left source, location
- ARG2: attribute of arg1

more

leave.02 - "give"

- ARG0: giver / leaver agent
- ARG1: thing given theme
- ARG2: benefactive / given-to location, recipient, beneficiary

more

leave.04 - "leave for"

Lexicon: What it doesn’t cover

- Sense disambiguation only applies to PropBank predicates (mostly events)!
- Other terms are not disambiguated. (Excluding named entities)

```
(o / obey-01
      :ARG0 (w / we)
      :ARG1 (l / law
            :topic (t / thermodynamics))
      :location (h / house
            :mod (t2 / this)))
```

Lexicon: what “want-01” and “:ARG0” mean

Not doing synonym sets: etymologically unrelated terms have different rolesets

fear-01	My fear of snakes
terrify-01	I’m terrified of snakes
creep_out.03	Snakes creep me out

- But we do generalize across parts of speech:

fear-01

My fear of snakes	Snakes terrify me
I am fearful of snakes	I’m terrified of snakes
I fear snakes	
I’m afraid of snakes	**terrify-01**
fear-01	

The man described the mission as a disaster.

The man’s description of the mission: disaster.

As the man described it, the mission was a disaster.

The man described the mission as disastrous.

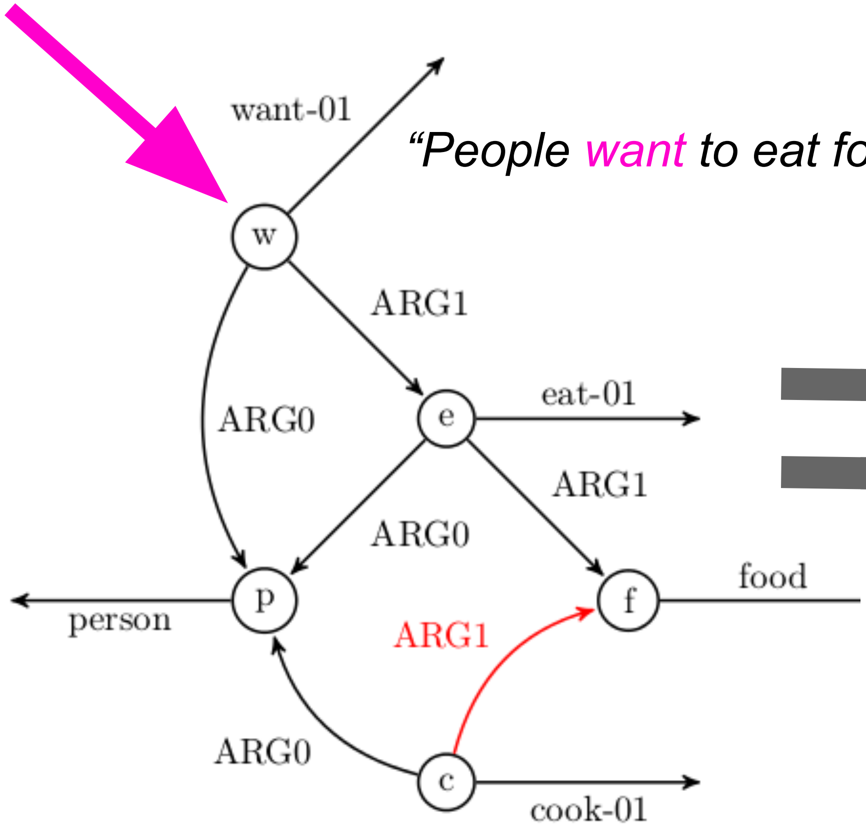
CANONICALIZE

```
(d / describe-01
    :ARG0 (m / man)
    :ARG1 (m2 / mission)
    :ARG2 (d / disaster))
```

Focus

- Which concept should go on top?
  - Conceptually, the main assertion of (the declarative version of) the sentence.
  - Linguistically, usually the main predication of the sentence.
- The concept on top is called the **focus**.
  - Must be a root (no incoming edges).

Focus

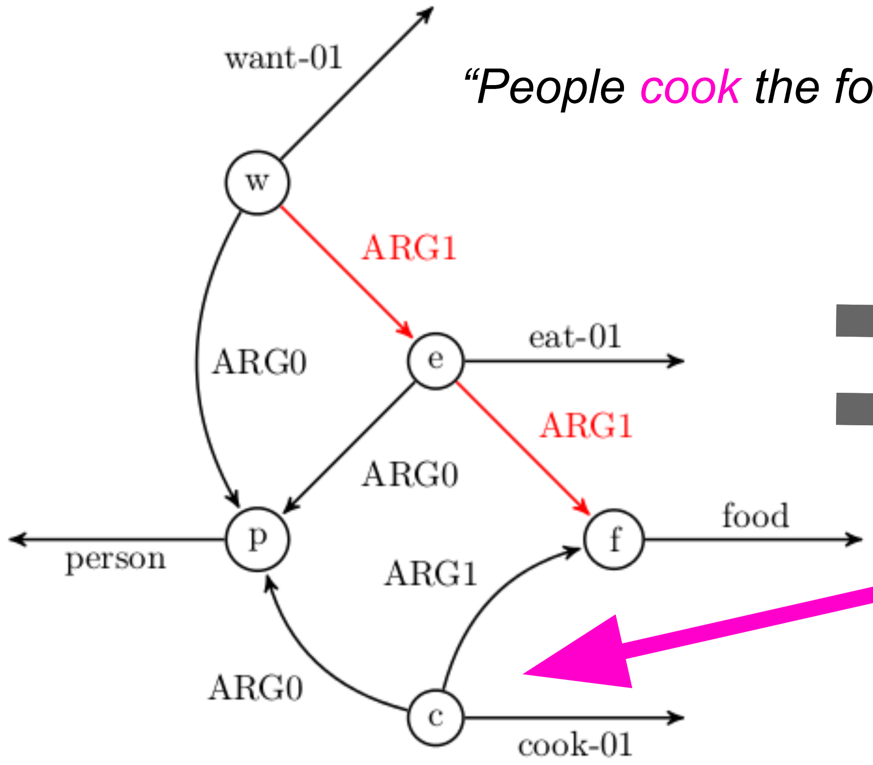


"People want to eat food that they cooked themselves"

=

```
(w / want-01
    :ARG0 (p / person)
    :ARG1 (e / eat-01
        :ARG0 p
        :ARG1 (f / food
            :ARG1-of (c / cook-01
                :ARG0 p))))
```

Focus



"People cook the food that they want to eat themselves"

=

```
(c / cook-01
      :ARG0 (p / person)
      :ARG1 (f / food
            :ARG1-of  (e / eat-01
                  :ARG0 p
                  :ARG1-of (w / want-01
                        :ARG0 p)))
```

Focus

“People want to eat food that they cooked themselves”

```
(w / want-01
    :ARG0 (p / person)
    :ARG1 (e / eat-01
        :ARG0 p
        :ARG1 (f / food
            :ARG1-of (c / cook-01
                :ARG0 p))))
```

“People cook the food that they want to eat themselves”

```
(c / cook-01
    :ARG0 (p / person)
    :ARG1 (f / food
        :ARG1-of  (w / want-01
          :ARG0 p
          :ARG1 (e / eat-01
                :ARG0 p)))
```

Propositionally, these are the same! But different emphasis.

Attribution of properties

Depending on focus, we use special roles `:mod` or `:domain` (these are inverses of each other).

the tasty food
There is tasty food.

```
(f / food
    :mod (t / tasty))
```

the tastiness of the food
The food is tasty.

```
(t / tasty
    :domain (f / food))
```

seeing the tasty food
seeing the food that is tasty

```
(s / see-01
    :ARG1 (f / food
                :mod (t / tasty)))
```

*seeing **that** the food is tasty*

```
(s / see-01
    :ARG1 (t / tasty
                :domain (f / food)))
```

Attribution of properties

Also for attributive/predicative demonstratives and nominals:

this house

```
(h / house
    :mod (t / this))
```

this is a house

```
(h / house
    :domain (t / this))
```

a monster truck

```
(h / truck
    :mod (m / monster))
```

the truck is a monster

```
(m / monster
    :domain (t / truck))
```

Non-core Roles

- Non-core arguments: not predicate-specific (not listed in lexicon)
- The boy wanted to go yesterday

```
(w / want-01
     :ARG0 (b / boy)
     :ARG1 (g / go-01
          :arg0 b)
          :time (y / yesterday))
```

Non-core Roles

- Relations that aren’t predicate-specific are handled with a large inventory of non-core semantic roles.

`:time`	`:location`	`:purpose`	`:frequency`
`:topic`	`:poss`	`:part`	`:manner`
...and many more! The full list is in the handout			

Non-core Roles

- There is also another kind of numbered argument for things where the number means nothing other than order: **:op#**
- Apples and bananas

```
(a / and
    :op1 (a2 / apple)
    :op2 (b / banana))
```

Non-core Roles

- There is also another kind of numbered argument for things where the number means nothing other than order: **:op#**
- Competition between lions, tigers and bears

```
(b / between
    :op1 (l / lion)
    :op2 (t / tiger)
    :op3 (b2 /bear))
```

Constants

String	Numeric

`name :op1 "Yoda"`
`:time "16:30"`
	`:quant 5`

Named	**+/-**

`monetary-quantity :unit dollar`
`:mode imperative`
	`:polarity -`
	`:polite +`

Constants vs. Concepts

- A **concept** is a type. For every concept node there will be ≥1 instance variable/node.
  - An instance can be mentioned multiple times.
  - Multiple instances of the same concept can be mentioned.
- **Constants** are singleton nodes: no variable, just a value. Specific non-core roles allow constant values.

Negation

I am not a crook.

```
(c / crook
     :domain (i / i)
     :polarity -)
```

Negation

Negation goes where it is logical:

I don’t believe we’ve met.

(*meaning:* ‘I believe we haven’t met.’)

```
(b / believe-01
    :ARG0 (i / i)
    :ARG1 (m / meet-02 :polarity -
             :ARG0 (w / we)))
```

Negation by morphology

an unhappy cat

```
(c / cat
    :mod (h / happy :polarity -))
```

illegible writing

```
(t / thing
   :ARG1-of (w / write-01
               :manner (l / legible :polarity -)))
```

Stemming plain concepts

- Non-event concepts are simply stemmed (drop plurality, articles)

cats
a cat
the cat
the cats

```
(c / cat)
```

Entities

- Proper names are represented with a `name` node

“Viacom”

```
(c / company
    :name (n / name :op1 “Viacom”))
```

“Barack Obama”

```
(p / person
    :name (n / name :op1 “Barack” :op2 “Obama”)
```

- The words of the name are string constants
- `<entity_type> :name <name_node>`

Entities

“Viacom”

```
(c / company
    :name (n / name :op1 “Viacom”))
    :wiki Viacom
```

“Barack Obama”

```
(p / person
    :name (n / name :op1 “Barack” :op2 “Obama”)
    :wiki Barack_Obama
```

- Wikification (`:wiki <page_name>`) in followup annotation pass

Entities

- An **ontology of entity types** is used *only if you do not have a more specific term in the sentence.*
- If a specific descriptor is present, we just use that word instead of finding the closest concept in the ontology.

"a Ford"

```
(v / vehicle
   :name (n / name :op1 "Ford"))
```

"a Ford truck"

```
(t / truck
    :name (n / name :op1 "Ford"))
```

- Ontology is large (100+ types) and hierarchical:

$$\begin{tikzcd}
 & \text{Location} \arrow[r] & \text{city; city-district; county; state; province ...} \\
\text{Thing} \arrow[ur] \arrow[dr] & & \\
 & \text{Product} \arrow[r] & \text{vehicle; ship; aircraft; aircraft-type...}
\end{tikzcd}$$

Entities

- There are also special entities that allow us to do very structured annotation of measurable quantities.
- **“Tuesday the 19th”   “five bucks”   “$3 / gallon”**

```
(d / date-entity
   :weekday Tuesday
   :day 19
```

```
(m / monetary-quantity
   :unit dollar
   :quant 5)
```

```
(r / rate-entity-91
         :ARG1 (m / monetary-quantity
                  :unit dollar
                  :quant 3 )
         :ARG2 (v / volume-quantity
                  :unit gallon
                  :quant 1 )
```

Temporal & Value Entities

- There are also special entities that allow us to do very structured annotation of measurable quantities.
- **“Tuesday the 19th”   “five bucks”   “$3 / gallon”**

```
(d / date-entity
   :weekday (t / tuesday)
   :day 19
```

```
(m / monetary-quantity
   :unit dollar
   :quant 5)
```

```
(r / rate-entity-91
      :ARG1 (m / monetary-quantity
            :unit dollar
            :quant 3)
      :ARG2 (v / volume-quantity
            :unit gallon
            :quant 1)
```

Designed to be similar to TIMEX normalization

Pronouns

- Pronouns with antecedents in the sentence are just reentrancies.
- Pronouns without antecedents in the sentence are just the pronoun (made nominative). “I” is lowercased.

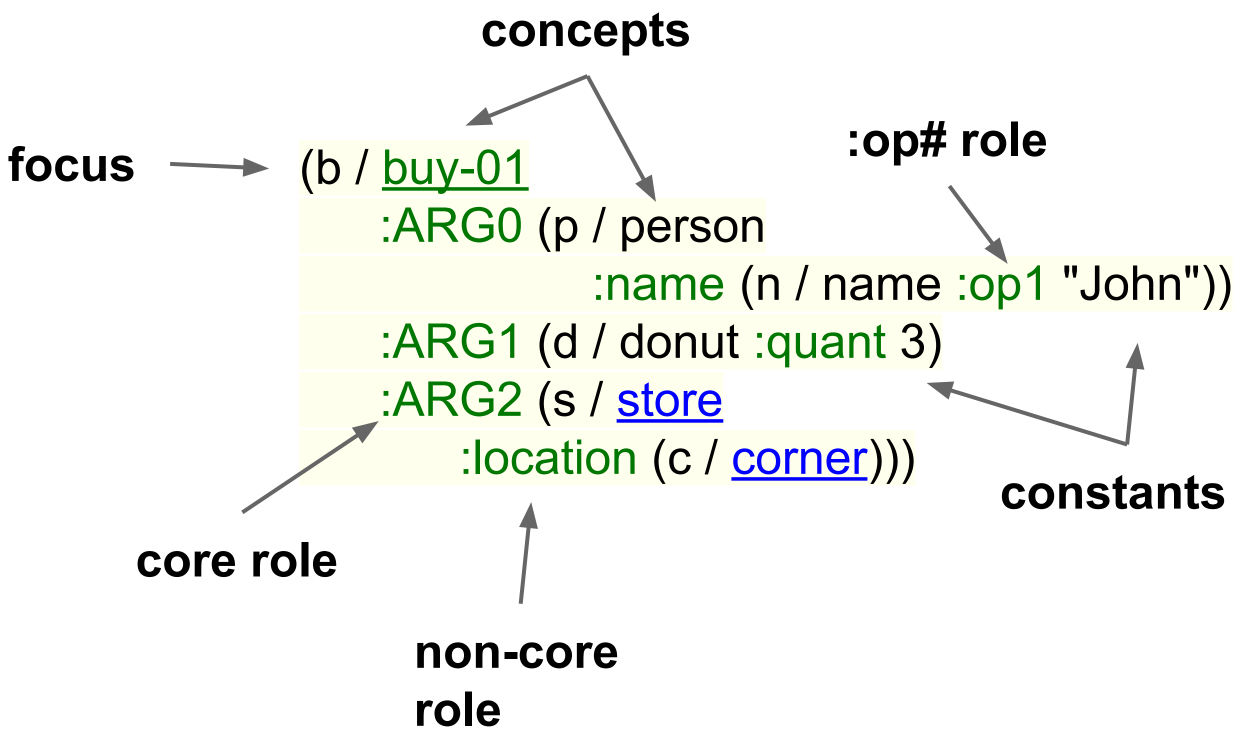
John$_i$ asked Mary to tutor him$_i$

```
(a / ask-02
      :ARG0 (p / person :name (n / name :op1 "John"))
      :ARG1 (t / tutor-01
            :ARG0 p2
            :ARG1 p)
      :ARG2 (p2 / person :name (n2 / name :op1 “Mary”)))
```

Mary was asked to tutor him

```
(a / ask-02
      :ARG1 (t / tutor-01
            :ARG0 p2
            :ARG1 (h / he))
      :ARG2 (p2 / person :name (n2 / name :op1
"Mary")))
```

So Remember...

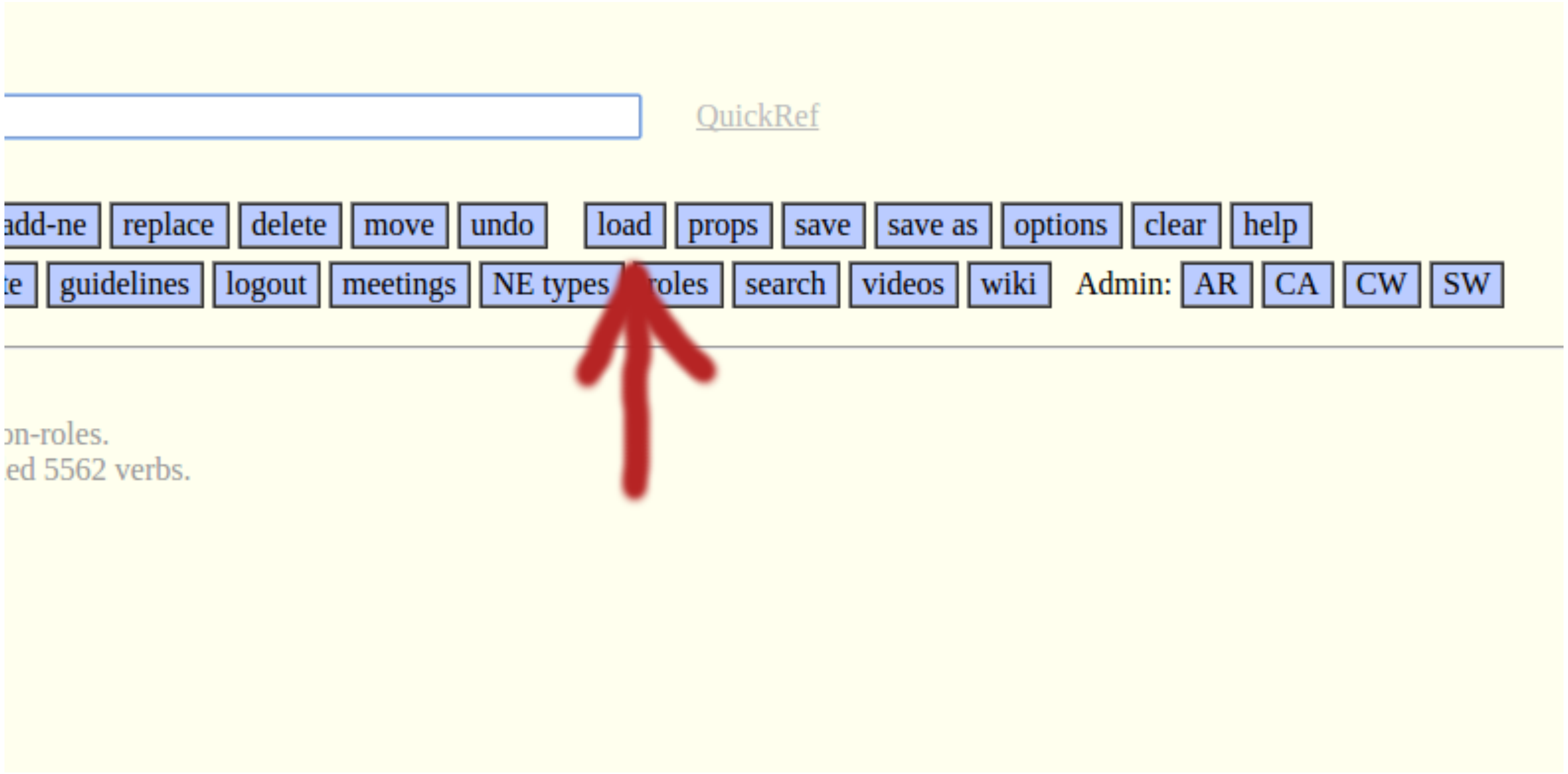


```
(b / buy-01
      :ARG0 (p / person
                  :name (n / name :op1 "John"))
      :ARG1 (d / donut :quant 3)
      :ARG2 (s / store
            :location (c / corner)))
```

Hands-On Review Examples!

http://tiny.cc/amreditor

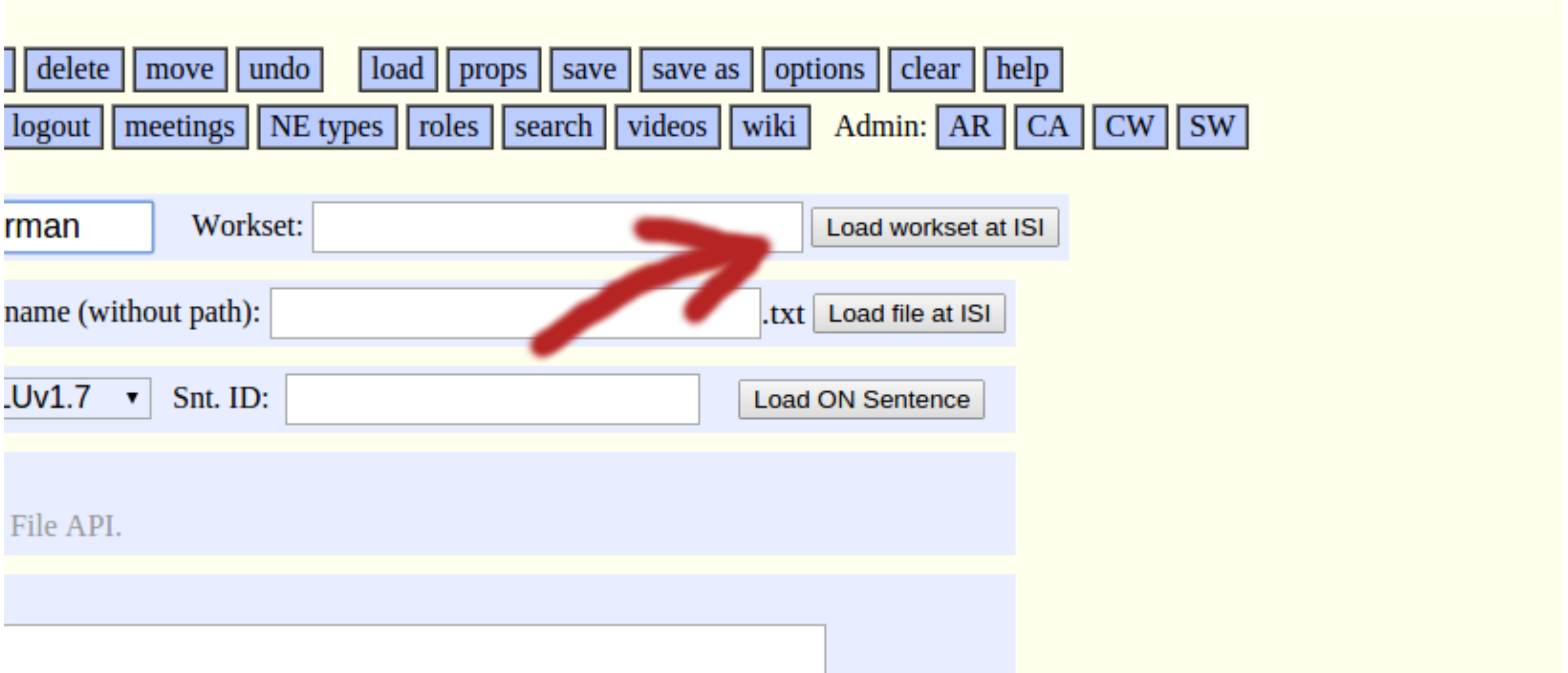
Hands-On Review Examples!



on-roles.
ed 5562 verbs.

Hands-On Review Examples!

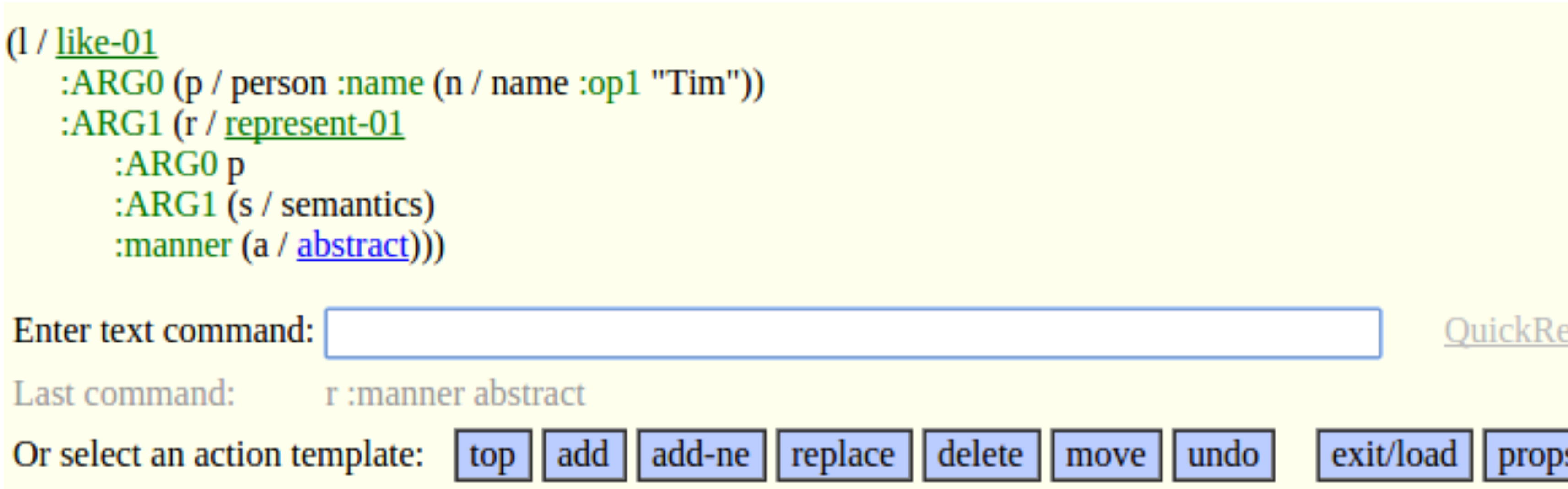
We are loading the “NAACL-tutorial” sentences.



Hands-On Review Examples!

“Tim likes to represent semantics abstractly”

We’ll walk slowly through a first sentence.



```
(l / like-01
    :ARG0 (p / person :name (n / name :op1 "Tim"))
    :ARG1 (r / represent-01
        :ARG0 p
        :ARG1 (s / semantics)
        :manner (a / abstract)))
```

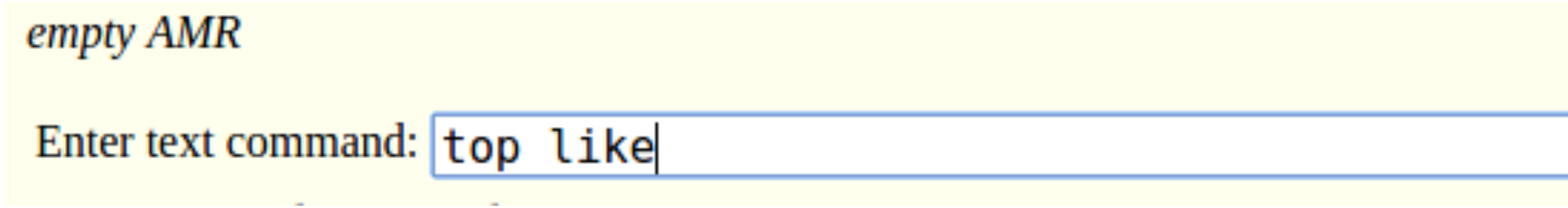
Enter text command:

Last command: r :manner abstract

Hands-On Review Examples!

“Tim likes to represent semantics abstractly”

“top” is how to make the root



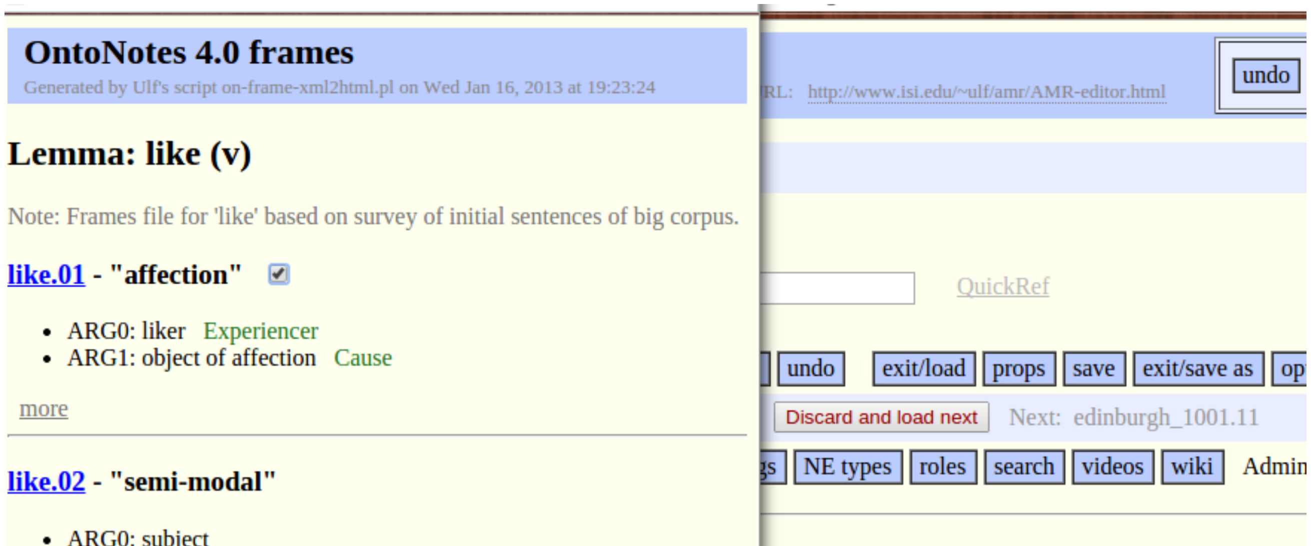
empty AMR

Enter text command: `top like`

Hands-On Review Examples!

“Tim likes to represent semantics abstractly”

click on “like” to see senses



OntoNotes 4.0 frames

Generated by Ulf's script on-frame-xml2html.pl on Wed Jan 16, 2013 at 19:23:24

Lemma: like (v)

Note: Frames file for 'like' based on survey of initial sentences of big corpus.

like.01 - "affection" ✓

- ARG0: liker Experiencer
- ARG1: object of affection Cause

more

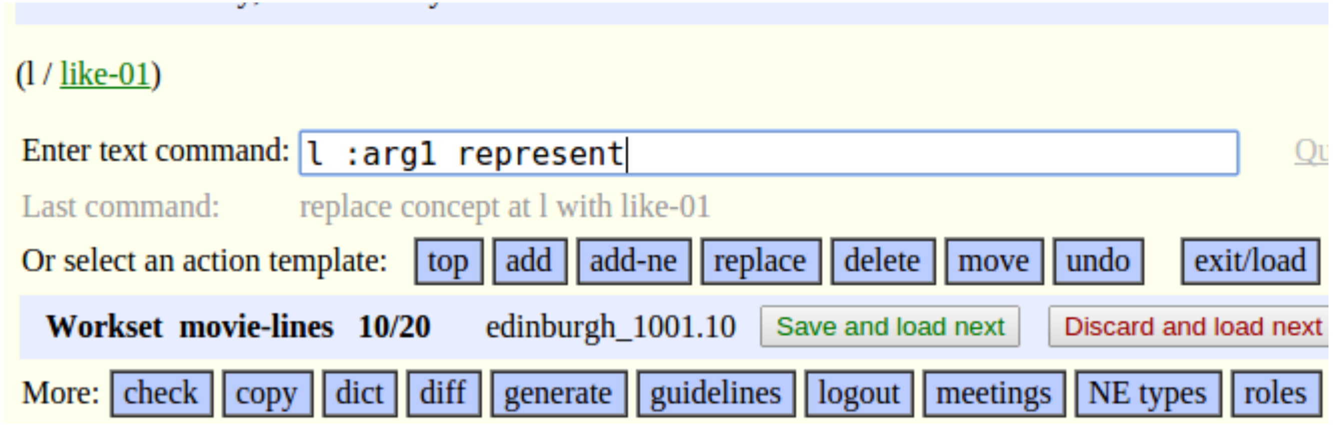
like.02 - "semi-modal"

- ARG0: subject

Hands-On Review Examples!

"Tim likes to represent semantics abstractly"

adding new relations: variable :role concept


(l / like-01)

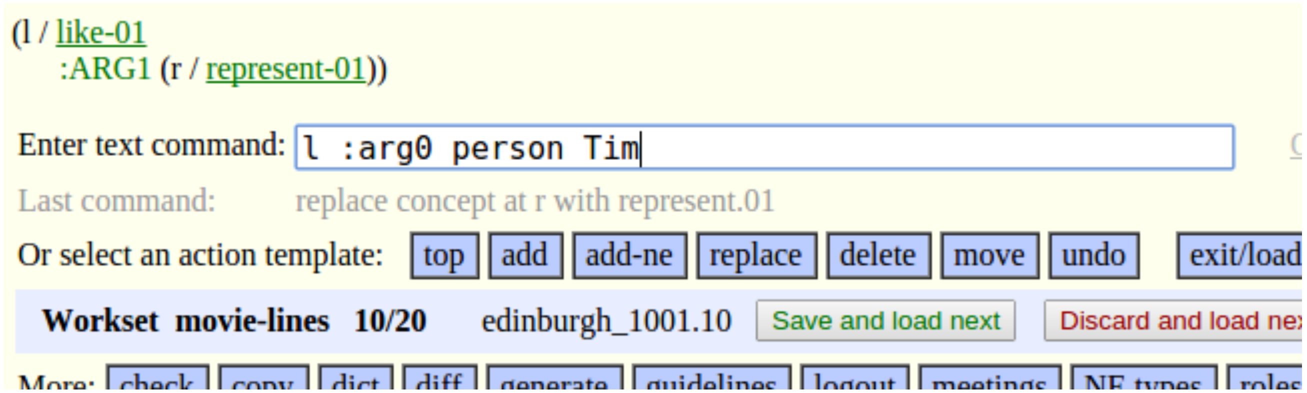
Enter text command: `l :arg1 represent`

Last command: replace concept at l with like-01

Hands-On Review Examples!

"Tim likes to represent semantics abstractly"

Everything after the third term is automatically added as a name



```
(l / like-01
    :ARG1 (r / represent-01))
```

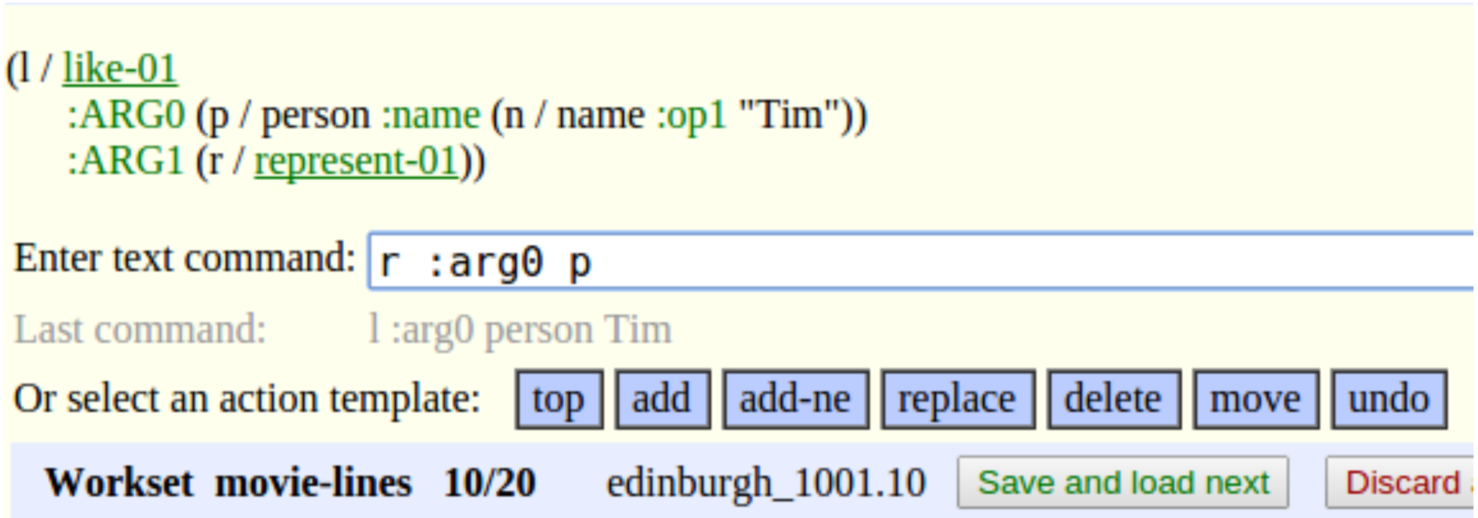
Enter text command: `l :arg0 person Tim`

Last command: replace concept at r with represent.01

Hands-On Review Examples!

"Tim likes to represent semantics abstractly"

Reentrancies are variable :role variable



```
(l / like-01
    :ARG0 (p / person :name (n / name :op1 "Tim"))
    :ARG1 (r / represent-01))
```

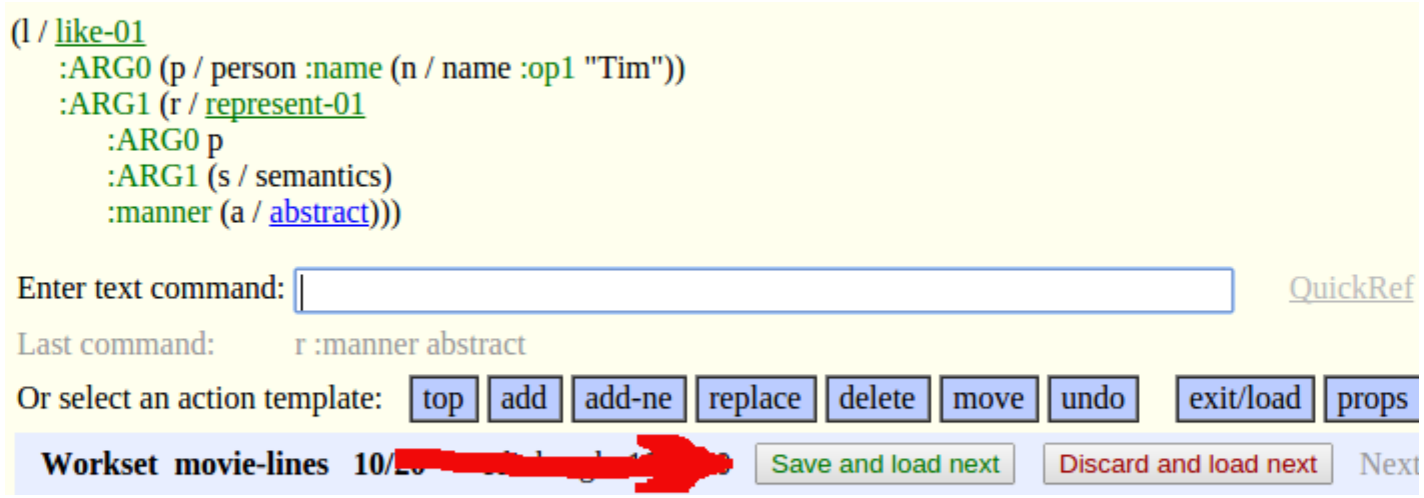
Enter text command: `r :arg0 p`

Last command: l :arg0 person Tim

Hands-On Review Examples!

“Tim likes to represent semantics abstractly”

Add the rest and “save and load next”



```
(l / like-01
    :ARG0 (p / person :name (n / name :op1 "Tim"))
    :ARG1 (r / represent-01
        :ARG0 p
        :ARG1 (s / semantics)
        :manner (a / abstract)))
```

Enter text command: QuickRef

Last command: r :manner abstract

Another hands-on example

“I hope Dumbledore likes my orange socks.”

Another hands-on example

“I hope Dumbledore likes my orange socks.”

Advanced Topics!

Decomposition into concepts

Other contexts for introducing concepts that don’t have words in the data: decomposing complex morphology.

A shoe salesman

```
(p / person
    :ARG0-of (s / sell-01
        :ARG1 (s2 / shoe)))
```

The Indian Government

```
(g / government-organization
    :ARG0-of (g2 / govern-01
        :ARG1 (c / country :name (n / name :op1 "India"))))
```

Decomposition into concepts

Other contexts for introducing concepts that don’t have words in the data: decomposing complex morphology.

Notice that we turn pertainyms like “Indian” into the entity they refer to

The Indian Government

```
(g / government-organization
      :ARG0-of (g2 / govern-01
            :ARG1 (c / country :name (n / name :op1 "India"))))
```

Throwing away light semantics

- Superficial syntactic patterns (light verbs, copular constructions, …) are canonicalized to a standard semantic form.

“John is nice”

```
(n / nice-41
        :arg1 (p/ person :name (n /name op1 “John”))
```

(We don’t allow “to be” at all!)

“John took a bath”

```
(b / bathe-01
    :arg0 (p / person :name (name :op1 “John”))
```

all light verbs convert to
the nearest verbal sense.

Reification

be-located-at-91 - "reification of :location"

ARG1: entity

ARG2: location

- *the man at the store*
  - (m / man :location (s / store))
- What about: *the man **always** at the store*?
  - Need to “modify” the relation!
  - Solution: Convert (“**reify**”) the relation w/ a special frame
  - (m / man

```
(m / man
      :ARG1-of (b / be-located-at-91
                   :ARG2 (s / store)
                   :time (a / always)))
```

Reification

- Reification also allows a relational predicate to be focused.
- *The man* ***is*** *at the store.*

  -
    ```
    (b / be-located-at-91 :ARG1 (m / man)
          :ARG2 (s / store))
    ```
- *I think the man is at the store*.

  -
    ```
    (t / think-01 :ARG0 (i / i)
          :ARG1 (b / be-located-at-91
                      :ARG1 (m / man) :ARG2 (s / store)))
    ```

Reification

- Every role has a designated reification—either a verb frame or a special -91 frame.
  - have-purpose-91, have-polarity-91, have-part-91, …
  - ~~have-topic-91~~ concern-02
- *These slides **are about** semantics.*

  -
    ```
    (c / concern-02
          :ARG0 (s / slide :mod (t / this))
          :ARG1 (s2 / semantics))
    ```

More special predicates

- Special predicates for individual/individual and individual/group relationships.
- *He's a pilot for TWA.*

  -
    ```
    (h / have-org-role-91
        :ARG0 (h / he)
        :ARG1 (c / company :name (n / name :op1 “TWA”))
        :ARG2 (p / pilot))
    ```

More special predicates

- Special predicates for individual/individual and individual/group relationships.
- *I am your father.*

  -
    ```
    (h / have-rel-role-91
        :ARG0 (i / i)
        :ARG1 (y / you)
        :ARG2 (f / father))
    ```

Copying

One word can result in multiple predicates

I ate a sandwich on Thursday and sushi on Friday.

```
(a / and
      :op1 (e / eat-01
            :ARG0 (i / i)
            :ARG1 (s / sandwich)
            :time (d / date-entity :weekday "Thursday"))
      :op2 (e2 / eat-01
            :ARG0 i
            :ARG1 (s2 / sushi)
            :time (d2 / date-entity :weekday "Friday")))
```

Set Operations

We have a predicate “include-91” for sets

I ate five of the 12 donuts” is processed as “I ate five donut out of a set of 12 donuts”

This is useful with our “set” predicate

“I ate 5 of the 12 donuts”

```
(e / eat-01
      :ARG0 (i / i)
      :ARG1 (d / donut :quant 5
            :ARG1-of (i2 / include-91
                  :ARG2 (d2 / donut :quant 12))))
```

include.91 - "subset"

ARG1: subset (or member)

ARG2: superset

ARG3: relative size of subset compared to superset

Set Operations

10% of smokers die of lung cancer.

```
(i / include-91
    :ARG1 (p / person
        :ARG1-of (d / die-01
            :ARG1-of (c2 / cause-01
                :ARG0 (c / cancer
                    :mod (l / lung)))))
    :ARG2 (p2 / person
        :ARG0-of (s2 / smoke-02))
    :ARG3 (p3 / percentage-entity :value 10))
```

Of the set of people who smoke….

… 10% of that set …

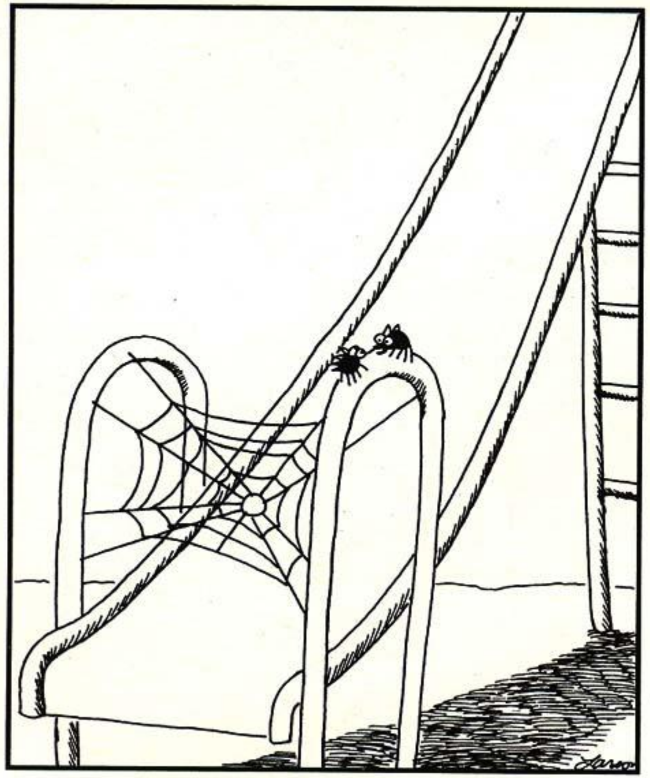
.. are people who die because of lung cancer

Many small additional patterns

- The AMR dictionary has conventions for many special cases.
- For example, “like” can be “resemble-01”:

If we pull this off, we’ll eat like kings

```
(e / eat-01
      :ARG0 (w / we)
      :ARG1-of (r / resemble-01
            :ARG2 (e2 / eat-01
                  :ARG0 (k / king)))
      :condition (p / pull-03
            :ARG0 w
            :ARG1 (t / this)))
```


“If we pull this off, we’ll eat like kings.”

Let’s look at real data

```
(n / need-01
      :ARG0 (w / we)
      :ARG1 (b / borrow-01
            :ARG0 w
            :ARG1 (p / percentage-entity :value 55
                  :ARG1-of (i / include-91
                        :ARG2 (p2 / price
                              :mod (h / hammer))))
            :time (u / until
                  :op1 (p3 / possible
                        :domain (g / get-01
                              :ARG0 w
                              :ARG1 (p4 / permit-01
                                    :ARG1 (p5 / plan-01)
                                    :purpose-of (r / restore-01)
                                    :ARG0-of (a / allow-01
                                          :ARG1 (m / mortgage-01
a.                                              :ARG0 w))))))))
```

We need to borrow 55% of the hammer price until we can get planning permission for restoration which will allow us to get a mortgage

Nearly identical AMRs:
we need a loan for 55% of the hammer price
of the full hammer price, we just need to borrow 55%

Let’s look at real data

```
(n / need-01
     :ARG0 (w / we)
     :ARG1 (b / borrow-01
          :ARG0 w
          :ARG1 (p / percentage-entity :value 55
               :ARG1-of (i / include-91
                    :ARG2 (p2 / price
                         :mod (h / hammer))))
          :time (u / until
               :op1 (p3 / possible
                    :domain (g / get-01
                         :ARG0 w
                         :ARG1 (p4 / permit-01
                              :ARG1 (p5 / plan-01)
                              :purpose-of (r / restore-01)
                              :ARG0-of (a / allow-01
                                   :ARG1 (m / mortgage-01
a.                                       :ARG0 w))))))))
```

We need to borrow 55% of the hammer price until we can get planning permission for restoration which will allow us to get a mortgage

identical AMRs
until such time as we get permission to plan for restoration
up until we get permits for restoration planning

Let’s look at real data

```
(n / need-01
    :ARG0 (w / we)
    :ARG1 (b / borrow-01
        :ARG0 w
        :ARG1 (p / percentage-entity :value 55
            :ARG1-of (i / include-91
                :ARG2 (p2 / price
                    :mod (h / hammer))))
        :time (u / until
            :op1 (p3 / possible
                :domain (g / get-01
                    :ARG0 w
                    :ARG1 (p4 / permit-01
                        :ARG1 (p5 / plan-01)
                        :purpose-of (r / restore-01)
                        :ARG0-of (a / allow-01
                            :ARG1 (m / mortgage-01
                                :ARG0 w))))))))
```

We need to borrow 55% of the hammer price until we can get planning permission for restoration which will allow us to get a mortgage

similar AMRs
permits allowing us to get a mortage

English Datasets

AMR Bank: *The Little Prince*
(novel; English translation)

LDC Releases
(news, discussion forums, etc.)

Data

- AMR Bank (Release 1.4; http://amr.isi.edu/download/amr-bank-v1.4.txt)
    - English translation of *The Little Prince*, freely downloadable
- AMR Public Release 1.0 (LDC2014T12): largest public release w/ 13,051 AMRs
- DEFT Release 3 (LDC2013E117): evaluation data in Flanigan et al 2014, Wang et al 2015.
- DEFT Release 4 (LDC2014E41): largest release w/ 18,779 AMRs total
- DEFT Release 5 (Sep. 2015) will include wikification, (pretty much) no directed cycles
- Small (100-AMR) sets of Czech and Chinese AMRs have been annotated.
- Vanderwende et al. (2015) data to appear: several languages, automatically converted from logical forms
- PropBank will soon all be converted to AMR style (mapping nominalizations to verbs, etc) and re-released.

Comparison - Semantic Roles

AMR: 70+ non-core roles, many verb-sense specific roles (up to 5 args/roleset, more than 10,000 rolesets)

FrameNet: large inventory of frame-specific roles

VerbNet: inventory of thematic roles

Groningen Meaning Bank: VerbNet inventory

Most others: small inventory of roles (agent, theme, etc.)

Comparison - Sense Lexicon

Groningen Meaning Bank: (automatic) WordNet synsets

FrameNet/UCCA: Mark senses by frame/script, not lemma

AMR /PropBank: coarse-grained senses (get high ITA)

Prague Dependency TB: valency lexicon rolesets

Most others: undisambiguated concepts as predicates

Comparison - Entities

AMR: Rich named entity ontology (100+ types), wikification

GALE/Ontonotes Annotations: 29 types, 64 subtypes

Groningen Meaning Bank: 7 NE types

Domain-specific (ACE/UMLS/etc.): rich; not all entities

Others: no entity typing

Comparison - Alignment with text

Deepbank; Groningen Meaning Bank: Semantics linked up to a theory of its derivation from syntax (HPSG; CCG)

PropBank, Semantic Treebank: grounded in PTB

Most others: Some link to words in sentence

AMR: No alignment to text (plan to release a few alignments)

Comparison - Logic/Scope/Entailments

Deepbank; Groningen Meaning Bank: Semantics grounds out in logical formalisms (DRT and MRS, respectively)

AMR entailment: linkage between its lexicon and VerbNet may allow rich decomposition

AMR scope: No scope of quantification

Comparison - Size and Quality

AMR: 18,779 sentences, goes beyond newswire, fully manual

Prague Dependency TB: WSJ in Czech and English, manual

Deepbank; Groningen Meaning Bank: Large; automatic parses with human correction/feedback.

UCCA: fully manual, 160k tokens

Rich semantic systems with little affiliated data: TMR, LCS, ...